M&G No. 60001.0286US01/305236.01

REPRESENTING SPELLING AND GRAMMATICAL ERROR STATE IN AN XML DOCUMENT

Related Applications

This patent application is a continuation-in-part application under 35 United States Code § 120 of United States Patent Application No. 10/187,060 filed on June 28, 2002, which is incorporated herein by reference. An exemplary schema in accordance with the present invention is disclosed beginning on page 11 in an application entitled "Mixed Content Flexibility," Serial No. _____, Docket No. 60001.0275US01, filed December 2, 2003, which is hereby incorporated by reference in its entirety.

Background of the Invention

Markup Languages have attained wide popularity in recent years. One type of markup language, Extensible Markup Language (XML), is a universal language that provides a way to identify, exchange, and process various kinds of data. For example, XML is used to create documents that can be utilized by a variety of application programs. Elements of an XML file have an associated namespace and schema.

In XML, a namespace is a unique identifier for a collection of names that are used in XML documents as element types and attribute names. The name of a namespace is commonly used to uniquely identify each class of XML document. The unique namespaces differentiate markup elements that come from different sources and happen to have the same name.

XML Schemata provide a way to describe and validate data in an XML environment. A schema states what elements and attributes are used to describe content in an XML document, where each element is allowed, what types of text contents are allowed within it and which elements can appear within which other elements. The use of schemata ensures that the document is structured in a consistent manner. Schemata

may be created by a user and generally supported by an associated markup language, such as XML. By using an XML editor, the user can manipulate the XML file and generate XML documents that adhere to the schema the user has created. XML documents may be created to adhere to one or more schemata.

Recently, some word processors have begun producing documents that are somewhat XML compatible. For example, some documents may be parsed using an application that understands XML. Many features of the word processor, however, are not stored within the XML file. For example, spelling and grammar state information is not stored within the XML file. What is needed is a way to store spelling and grammar state within an XML document.

Summary of the Invention

The present invention is directed towards marking spelling and grammar errors and proofing state of a word-processing document within XML.

According to one aspect of the invention, markers are used to show where a spelling or grammar error has occurred within the document.

According to another aspect of the invention, the proof state of the document is also stored. For example, has the document been fully checked for spelling or grammatical errors.

According to another aspect of the invention, the word processing document stored as XML may be parsed by an application that understands XML even though it is not the creator of the document.

Brief Description of the Drawings

FIGURE 1 illustrates an exemplary computing device that may be used in one exemplary embodiment of the present invention;

FIGURE 2 is a block diagram illustrating an exemplary environment for practicing the present invention;

FIGURE 3 illustrates an exemplary ML file;

FIGURE 4 illustrates an exemplary ML file including markers for a grammar error;

FIGURE 5 illustrates an exemplary ML file including markers for a spelling error; and

FIGURE 6 illustrates an exemplary definition for a proof state element, in accordance with aspects of the invention.

Detailed Description of the Preferred Embodiment

Throughout the specification and claims, the following terms take the meanings explicitly associated herein, unless the context clearly dictates otherwise.

The terms "markup language" or "ML" refer to a language for special codes within a document that specify how parts of the document are to be interpreted by an application. In a word-processor file, the markup language specifies how the text is to be formatted or laid out, whereas in a particular customer schema, the ML tends to specify the text's meaning according to that customer's wishes (e.g., customerName, address, etc. The ML is typically supported by a word-processor and may adhere to the rules of other markup languages, such as XML, while creating further rules of its own.

The term "element" refers to the basic unit of an ML document. The element may contain attributes, other elements, text, and other building blocks for an ML document.

The term "tag" refers to a command inserted in a document that delineates elements within an ML document. Each element can have no more than two tags: the start tag and the end tag. It is possible to have an empty element (with no content) in which case one tag is allowed.

The content between the tags is considered the element's "children" (or descendants). Hence other elements embedded in the element's content are called "child elements" or "child nodes" or the element. Text embedded directly in the content of the element is considered the element's "child text nodes". Together, the child elements and the text within an element constitute that element's "content".

The term "attribute" refers to an additional property set to a particular value and associated with the element. Elements may have an arbitrary number of attribute settings associated with them, including none. Attributes are used to associate additional information with an element that will not contain additional elements, or be treated as a text node.

Illustrative Operating Environment

With reference to FIGURE 1, one exemplary system for implementing the invention includes a computing device, such as computing device 100. In a very basic configuration, computing device 100 typically includes at least one processing unit 102 and system memory 104. Depending on the exact configuration and type of computing device, system memory 104 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory 104 typically includes an operating system 105, one or more applications 106, and may include program data 107. In one embodiment, application 106 may include a word-processor application 120 that further includes spelling and grammar application 122. This basic configuration is illustrated in FIGURE 1 by those components within dashed line 108.

Computing device 100 may have additional features or functionality. For example, computing device 100 may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in FIGURE 1 by removable storage 109 and non-removable storage 110. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory 104, removable storage 109 and non-removable storage 110 are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired

information and which can be accessed by computing device 100. Any such computer storage media may be part of device 100. Computing device 100 may also have input device(s) 112 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 114 such as a display, speakers, printer, etc. may also be included. These devices are well know in the art and need not be discussed at length here.

Computing device 100 may also contain communication connections 116 that allow the device to communicate with other computing devices 118, such as over a network. Communication connection 116 is one example of communication media. Communication media may typically be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

Word-Processor File Structure and Spelling and Grammar

Many word processors have built in spelling and grammar checking to assist users while editing their files. These are extremely powerful and popular features.

Storing the spelling and grammar state as XML, helps to make it possible to use other tools to look at the documents and tell the state of that document. Additionally, storing the states as XML helps to improve performance when opening the files using the word processor. If all the spelling and grammar errors are already marked, then the word processor does not have to pass over the entire document to check for spelling & grammar errors every time it is opened. It only needs to examine the areas that the user changes.

FIGURE 2 is a block diagram illustrating an exemplary environment for practicing the present invention. The exemplary environment shown in FIGURE 2 is a

word-processor environment 200 that includes word-processor 120, ML file 210, ML Schema 215, and ML validation engine 225.

In one embodiment, word-processor 120 has its own namespace or namespaces and a schema, or a set of schemas, that is defined for use with documents associated with word-processor 120. The set of tags and attributes defined by the schema for word-processor 120 define the format of a document to such an extent that it is referred to as its own native ML. Word-processor 120 internally validates ML file 210. When validated, the ML elements are examined as to whether they conform to the ML schema 215. A schema states what tags and attributes are used to describe content in an ML document, where each tag is allowed, and which tags can appear within other tags, ensuring that the documentation is structured the same way. Accordingly, ML 210 is valid when structured as set forth in arbitrary ML schema 215.

ML validation engine 225 operates similarly to other available validation engines for ML documents. ML validation engine 225 evaluates ML that is in the format of the ML validation engine 225. For example, XML elements are forwarded to an XML validation engine. In one embodiment, a greater number of validation engines may be associated with word-processor 120 for validating a greater number of ML formats.

FIGURE 3 illustrates an exemplary ML file in accordance with aspects of the present invention. ML file 300 includes ML elements. An element in a markup language usually includes an opening tag (indicated by a "<" and ">"), some content, and a closing tag (indicated by a "</" and ">"). In this example, tags associated with ML include a "me:" within the tag (e.g., 302). The “me:” prefix is used as shorthand notation for the namespace associated with the element.

There are enough ML elements for an application that understands XML to fully recreate the document from a single XML file. Hint tags may also be included that provide information to an application to help understand the content of the file. The text contained within the document follows the "text" tag, making it easy for an application to extract the text content from a word-processing document created in accordance with aspects of the invention.

Take an XML file that has the following sentence containing a grammatical error:

"I do be watching this."

Also assume that the above sentence has an XML element <me:XML> applied to it around "be watching", such that the output XML looks as illustrated in FIGURE 3.

Given that the example shown is valid, ML file 210 produces a document with a paragraph that includes the text "I do be watching this" in the first paragraph.

The phrase "do be" should be flagged as a grammar error. In order to maintain well formedness, the XML tag representing the error start can not be placed outside of the <me:XML> tag and end inside of it. For example, the following is not well formed XML:

```
<para>
    <run>
        <text>I</text>
    </run>
    <GrammarError>
        <run>
            <text>do</text>
        </run>
    <me:XML>
        <run>
            <text>be</text>
        </run>
    </GrammarError>
        <run>
            <text> watching</text>
        </run>
```

```
</me:XML>
<run>
        <text>this</text>
</run>
</para>
```

FIGURE 4 illustrates an exemplary ML file including markers for a grammar error, in accordance with aspects of the invention. As illustrated in FIGURE 4, the ML file includes a single tag (402) to represent the start of the grammar error, and another single tag (404) to represent the end of the grammar error. According to one emobodiment of the invention, the tag representing the start of the grammar error is **<proofErr type="GrammarStart"/>** and the tag representing the end of the grammar error is **<proofErr type="GrammarEnd"/ >**. Other tags may be used.

The following list is an exemplary list of **proofErr** types:

```
<xsd:simpleType name="proofErrType">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="spellStart"></xsd:enumeration>
                <xsd:enumeration value="spellEnd">
                        <xsd:annotation>
                                <xsd:documentation>The spelling error is
contained within spellStart and spellEnd.</xsd:documentation>
                        </xsd:annotation>
                </xsd:enumeration>
                <xsd:enumeration value="gramStart"></xsd:enumeration>
                <xsd:enumeration value="gramEnd">
                        <xsd:annotation>
                                <xsd:documentation>The grammar error is
contained within gramStart and gramEnd.</xsd:documentation>
                        </xsd:annotation>
                </xsd:enumeration>
        </xsd:restriction>
```

</xsd:simpleType>

FIGURE 5 illustrates an exemplary ML file including markers for a spelling error, in accordance with aspects of the invention. As illustrated in FIGURE 5, the ML file includes a single tag (502) to represent the start of the spelling error, and another single tag (504) to represent the end of the spelling error.

FIGURE 6 illustrates an exemplary definition for a proof state element, in accordance with aspects of the invention. The proof state of the document indicates whether the document has been fully checked for spelling & grammar errors. The proof state helps to make it possible to ignore a spelling error and save. When the file is reopened, the spelling error is still ignored. If proof state did not exist, the word processor would not know if the document has already been checked for spelling mistakes, and would therefore re-flag the error.

According to one embodiment, the ignored spelling error is not marked, and would, therefore, be re-flagged if there were no state information. A spelling error is explicitly marked with XML, but an ignored error has no marking. Any misspelling that is not flagged (when the proof state is clear), can be assumed to have already been ignored by the user.

The following is a commented definition for the **proofState** element that defines the state of the document:

```
<xsd:element name="proofState" type="proofProperty" minOccurs="0">
    <xsd:annotation>
        <xsd:documentation>The state of proofing tools in this
document.</xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:complexType name="proofProperty">
    <xsd:annotation>
        <xsd:documentation>The state of proofing tools in this
document.</xsd:documentation>
```

```
            </xsd:annotation>
            <xsd:attribute name="spelling" type="proofType" use="optional">
                <xsd:annotation>
                    <xsd:documentation>The state of the spell checker in this
document.</xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="grammar" type="proofType" use="optional">
                <xsd:annotation>
                    <xsd:documentation>The state of the grammar checker in this
document.</xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:complexType>

        <xsd:simpleType name="proofType">
            <xsd:annotation>
                <xsd:documentation>Proofing Tools state values.</xsd:documentation>
            </xsd:annotation>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="clean">
                    <xsd:annotation>
                        <xsd:documentation>The proofing tool finished checking
this document.  Errors are marked and only the errors will be rechecked on
open.</xsd:documentation>
                    </xsd:annotation>
                </xsd:enumeration>
                <xsd:enumeration value="dirty">
                    <xsd:annotation>
```

<xsd:documentation>The proofing tool did not finish checking this document. The entire document will be rechecked on open.</xsd:documentation>

</xsd:annotation>

</xsd:enumeration>

</xsd:restriction>

</xsd:simpleType>

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.